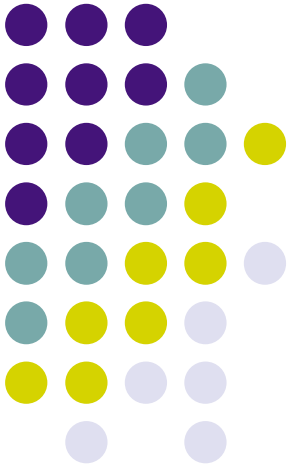
Miscellaneous Items

Tom Clune

SIVO Fortran 2003 Series

March 11, 2008



Logistics

- Materials for this series can be found at http://modelingguru.nasa.gov/clearspace/docs/DOC-1375
  - Contains slides and source code examples.
  - Latest materials may only be ready at-the-last-minute.
- Please be courteous:
  - Remote attendees should use “*6” to toggle the mute. This will minimize background noise for other attendees.
- Webex - under investigation

Outline

- Computing Environment
  - IO seen before
  - Count, get
- Array Constructor Syntax
- Module enhancements
  - IMPORT Statement
  - New attributes
    - PROTECTED
    - VOLATILE
  - Renaming operators
- Changes to intrinsic functions
- Length of names/statements
- Complex constants C = (0.0,pi)
- Support for international character sets

Computing Environment

- From the *intrinsic* module `ISO_FORTRAN_ENV`
- For the following assume we have launched the executable with the command line: % foo.x apple 5 z
- `COMMAND_ARGUMENT_COUNT()`
  - Returns integer number of command arguments
  - Example command returns 3
- `GET_COMMAND([COMMAND,LENGTH,STATUS])`
  - All `INTENT(OUT)` and `OPTIONAL`
  - `LENGTH` - integer # of characters in command
  - `STATUS` - integer (success/failure)
  - Results for example command:
    - COMMAND="foo.x apple 5 z"
    - LENGTH=15

Computing Env (cont'd)

- `GET_COMMAND_ARGUMENT(NUMBER[,VALUE,LENGTH,STATUS])`
  - NUMBER - selects argument
  - VALUE - character, intent(out) value of argument
  - LENGTH - number of characters in argument
  - STATUS - integer (success/failure)
  - Example command yields:
    - GET_COMMAND_ARGUMENT(0,VALUE,LENGTH) yields VALUE="foo.x", LENGTH=5
    - GET_COMMAND_ARGUMENT(2,VALUE,LENGTH) yields VALUE="5", LENGTH=1

Computing Env. (cont'd)

- `GET_ENVIRONMENT_VARIABLE(NAME[,VALUE, LENGTH,STATUS,TRIM_NAME])`
  - NAME - character, intent(in), name of environment variable
  - VALUE - character, intent(out) value of env variable
  - LENGTH - number of characters in value - 0 if does not exist
  - STATUS - success/fail
  - TRIM_NAME - logical, intent(in) for ignoring trailing blanks

Environment examples

- Getting command arguments:

```
use ISO_FORTRAN_ENV
character(len=MAXLEN_ARG) :: arg1, arg2
call get_command_argument(1, VALUE=arg1)
call get_command_argument(2, VALUE=arg2)
read(arg1,'(i)') nx
read(arg2,'(i)') ny
```

- Getting an environment variable:

```
use ISO_FORTRAN_ENV
character(len=100) :: myShell
call get_environment_variable('SHELL', myShell)
```

Array Constructor Syntax

- Can now use “[“ and “]” rather than “(/”, “/)” to construct arrays:
  x(1:5) = [0.,1.,2.,3.,4.]
- Can also specify type ***inside*** constructor
  - Follows rules of instrinsic assignment
  - Allows type conversion within the constructor
  - Convenient for mixing types/kinds/lengths
    - Mixed real/integer: x(1:5) = [real :: 0,1.,2.,3,4]
    - Mixed string lengths:
      names = [character(len=10):: ‘SpongeBob’, ‘Patrick’]
  - Also useful for derived types:
    list = [myType :: a, b, c]

IMPORT Statement

- A common pitfall when using F90/F95 is the declaration of an interface block than needs to “use” a derived type defined in the same module:

```
module foo
    type bar
      integer :: I,J
    end type bar

    interface
       subroutine externFunc(B)
          use foo, only: bar ! Not allowed?
          type (bar) :: B
       end subroutine externFunc
    end interface
  ….
```

IMPORT Statement (cont'd)

- IMPORT is a new statement to address this issue
  - Very similar to USE statement
  - Specifies all entities in host scoping unit that are accessible
    - Use "ONLY" clause to limit selection
    - All entities are accessible by default
  - *Only* allowed in an interface body within a module
- Example:

```
…
interface
    subroutine externFunc(B)
       import foo, only: bar
       type (bar) :: B
    end subroutine externFunc
end interface
```

PROTECTED Attribute

- F2003 introduces the new attribute `PROTECTED` which provides a safety mechanism analogous to `INTENT(IN)`
  - Specifies that the variable (or pointer status) may be altered only *within* the host module
  - Property is recursive. I.e. if a variable of derived type is PROTECTED, all of its subjobjects also have the attribute
  - For pointers, only the *association* status is protected. The target may be modified elsewhere.

  - Example:

```
module foo
  private ! Good default
  real, public :: pi
  protected :: pi ! Allow value to be read
 …
```

VOLATILE Attribute

- Introduced for a data object to indicate that its value might be modified by means external to the program.
  - Non standard extensions (e.g. threads)
  - Card connected to external lab instrument
  - Etc.
- Effect is that the compiler is required to not rely on values in cache or other temporary memory.
  - Can prevent some common optimizations
- If an object has the `VOLATILE` attribute, so do all of its subobjects.
- For pointers, attribute refers only to the association status, *not* the target.

Renaming operators

- F2003 extends the rename capability on `USE` statements to include renaming operators that are not intrinsic operators:

```
USE MY_MODULE, OPERATOR(.MyAdd.) => OPERATOR(.ADD.)
```

  - This allows .MyAdd. to denote the operator .ADD. accessed from the module.

Changes to Intrinsic Functions

- Argument `COUNT_RATE` for `SYSTEM_CLOCK()` can now be of type real.
  - Previously had to convert integer to compute reciprocal to determine elapsed time
- `MAX, MAXLOC, MAXVAL, MIN, MINLOC, MINVAL` have all been extend to apply to type `CHARACTER`
- ATAN2, LOG, and SQRT have minor changes to take into account positive/negative zero for vendors that support the distinction.

Lengths of Names/Constants

- Variables may be declared with names of up to 63 characters
- Statements of up to 256 lines are permitted.
- Primarily aimed at supporting automatic code generation

Complex constants

- Named constants may be used to specify real or imaginary parts of a complex constant:

```
REAL, PARAMETER :: pi = 3.14159265358979323384
COMPLEX :: C = (0.0,pi)
```

Pitfalls and Best Practices

- Environment
  - Use LENGTH keyword to ensure buffers are large enough
  - Check status - did the command succeed?
- Use named constants when possible

Supported Features

Compiler	Ifort 9.1.049	Ifort 10.1	NAG 5.1	XLF 11.0	G95 0.90	Gfortran 20070810	pgi 6.2.4
Environment	no	yes	yes	yes	yes	yes	no
Array Constructor	yes*	yes*	yes*	yes	yes*	yes	no
Import	yes	yes	yes	yes	yes	yes	no
Protected	yes	yes	yes	yes	yes	yes	no
Volatile	yes	yes	yes	yes	yes	yes	no
Real clock_rate	no	no	no	yes	no	no	no
Complex constructor	yes	yes	no	yes	yes	yes	yes
Character max, min, etc	no	no	no	yes	yes	no	yes

Feel free to contribute if you have access to other compilers not mentioned!

Resources

- **SIVO Fortran 2003 series:** https://modelingguru.nasa.gov/clearspace/docs/DOC-1390
- **Questions to Modeling Guru:** https://modelingguru.nasa.gov
- **SIVO code examples on Modeling Guru**
- **Fortran 2003 standard:** http://www.open-std.org/jtc1/sc22/open/n3661.pdf
- ***John Reid summary:***
  - ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.pdf
  - ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.ps.gz
- ***Newsgroups***
  - http://groups.google.com/group/comp.lang.fortran
- ***Mailing list***
  - http://www.jiscmail.ac.uk/lists/comp-fortran-90.html

Next Fortran 2003 Session

- Introduction to Object Oriented Programming for Scientists
- Tom Clune will present
- Tuesday, April 08, 2008
- B28-E210 @ 12:00 noon